DSTO

AR-009-478

DSTO-RR-0070

XIsabelle: A Graphical User Interface to the Isabelle Theorem Prover

A. Cant and M. A. Ozols

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

XIsabelle: A Graphical User Interface to the Isabelle Theorem Prover

A. Cant and M.A. Ozols

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-RR-0070

ABSTRACT

Interactive theorem provers such as Isabelle are powerful tools, but are difficult and time-consuming to learn. If a suitable Graphical User Interface (GUI) is provided for such a tool, it can speed up the learning process considerably, leading to greater productivity for users of the tool, and increased takeup in industry. In this paper, we discuss the user-interface aspects of Isabelle, and formulate requirements for a GUI. XIsabelle, a GUI for Isabelle, is described in detail. XIsabelle uses standard, easily available, methods for providing X Windows wrappers to interactive non-GUI programs, namely Tcl/Tk and the program Expect.

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury. South Australia, Australia 5108

Telephone: (08) 259 6700
Fax: (08) 259 5980

AR-009-478
December 1995

XIsabelle: A Graphical User Interface to the Isabelle Theorem Prover

Executive Summary

The ability to carry out formal reasoning is vital in the development and analysis of *critical systems*, i.e. systems in which incorrect operation is unacceptable for safety, security, or other reasons. Formal methods (i.e. formal specification and verification) are mandated in some standards for safety-critical and security-critical software.

The task of carrying out formal proofs cannot be completely automated: in general the user must carry out the proof as far as the broad strategy is concerned. Users of proof assistants want their tools to be easy to use, as well as being powerful enough to prove results of interest. If this is done, it will result in increased industrial takeup, leading to increased assurance of safety or security in critical systems. It will also increase the user base, resulting in useful feedback and improvements to the tool. To achieve this, theorem provers need Graphical User Interfaces (GUIs).

In our work, we have been motivated by the need to provide support for two basic kinds of user. The first of these is the *verifier*, a verification expert who has detailed knowledge of the capabilities of a given theorem prover, and can use them to advantage to solve a particular problem. The other kind of user is the *engineer*, who is an expert in a particular application domain, but who does not have theorem proving expertise.

The aim of this paper is to describe the ideas behind XIsabelle, a Graphical User Interface for the theorem prover Isabelle. XIsabelle is written using the scripting language Tcl/Tk.

XIsabelle is more than just a plain graphical 'front-end' to Isabelle. It provides a number of features not directly accessible in Isabelle. Here are the main features:

- It permits theory browsing, i.e. inspecting the structure of a given logical theory.
- It largely removes the need for the user to learn Isabelle commands.
- It allows the user to carry out a proof by 'clicking'.
- It provides advice to the user on possible proof steps.
- Proofs can be saved and replayed at a later stage.

We believe that XIsabelle offers a most promising way ahead for an Isabelle GUI. Our aim is to stimulate discussion on the user-interface aspects of what is already a very powerful theorem-proving environment.

Authors

A. Cant
Electronics and Surveillance Research Laboratory
Defence Science and Technology Organisation

Tony Cant is a Senior Research Scientist within the Trusted Computer Systems Group. He leads a section which carries out research into formal methods and tools for safety and security critical systems, focusing on approaches to machine-assisted reasoning about their critical properties. He is also involved in policy and standards formulation for the procurement of safety critical systems for the Department of Defence. Tony joined DSTO in 1990. Prior to that, he held a number of academic research positions in mathematical physics, as well as working in science policy.

M. A. Ozols
Electronics and Surveillance Research Laboratory
Defence Science and Technology Organisation

Maris Ozols is a Research Scientist within the Trusted Computer Systems Group. His research interests include theoretical methods and tools for trusted system design, development and evaluation. Prior to joining DSTO in 1990, Maris completed a BSc (Hons) at the University of Adelaide in 1986, followed by mathematical research in algebra at Monash University, obtaining his PhD in 1991.

Contents

List of Tables

List of Figures

1 Introduction

The ability to carry out formal reasoning is vital in the development and analysis of *critical systems*, i.e. systems in which incorrect operation is unacceptable for safety, security, or other reasons. Formal methods (i.e. formal specification and verification) are being used in a number of projects [1], and are mandated in certain standards for safety-critical software, such as the UK Interim Defence Standards 00-55 and 00-56, and the Australian Ordnance Council's recent Pillar Proceeding [2, 3, 4]; and in standards for security-critical systems, for example the US DoD's Trusted Computer System Evaluation Criteria [5].

The construction of formal proofs is both intellectually challenging, and tedious in its details. A number of *proof assistants* have been developed which are intended to aid in the construction of formal proofs and, most importantly, to ensure that they are sound. Proof assistants such as verification tools and automated theorem provers play a crucial role in the development and analysis of such critical systems. They are also of considerable interest to mathematicians and logicians.

The task of carrying out formal proofs cannot be completely automated: in general the user must carry out the proof as far as the broad strategy is concerned. Theorem provers need to be *interactive.* The proof assistant's role is to provide a means to automate the process of building up proof steps from primitive ones: it is essentially a 'bookkeeping' and proof management role. This highlights the importance of the way that the user interacts with a theorem prover. Users of proof assistants want their tools to be easy to use, as well as being powerful enough to be able to prove results of interest.

Most proof assistants have simple command-line interfaces. There are a number of interactive tactical theorem provers, such as HOL [6] and Isabelle [7]. These provers are powerful and flexible tools. They have simple command-line interfaces, but the command language is extremely expressive, being a full functional programing language.

There is a heavy price to pay in using proof assistants. The command language takes a considerable time for a beginner to learn. Even expert users spend considerable time on 'bookkeeping' tasks when they would rather focus on carrying out proofs. On the other hand, such tools repay the user prepared to spend the effort with the ability to model and reason about a large range of problems.

However, the slow learning time limits the user base to those prepared to study the tools. In particular, the industrial takeup of these tools is restricted to companies prepared to make a long-term investment of time and effort. Widespread industrial takeup will only occur for tools seen as being sufficiently robust and long-lived [8].

For example, the Isabelle theorem prover [7] is especially powerful and versatile. It has been used by the authors to implement an environment for reasoning about formal language definitions via operational semantics, in particular for Standard ML programs [9]. Isabelle's use has mostly been limited to a relatively small number of academic sites, although it has recently been used to carry out a formal analysis of an emergency shutdown system [10]. The HOL system [6] is more widely-established, with a fairly large user base, and some industrial applications. ICL's version of the HOL system is a commercially available tool. However, HOL is not routinely used in industry.

It is essential to make automated theorem provers easier to use, without sacrificing their flexibility and power. This will increase their industrial takeup, leading to increased assurance of safety or security in critical systems. It will also increase the user base, resulting in useful feedback and improvements to the tool. To do this, such provers need a Graphical User Interface (GUI) which allows the user to begin to use the theorem prover without being distracted by matters of detail, and to reach the stage of doing real proofs as soon as possible.

A proof assistant which does have a well integrated graphical user interface (GUI) is the Mural system [12]. From conception, and throughout design, the developers have taken great care to provide much support to the users. However, although Mural has an impressive graphical interface, it is less powerful in some respects than the theorem provers mentioned above.

The need for such interfaces to existing command-line provers has already been partially addressed. Donald Syme has recently announced a Tcl/Tk-based interface to HOL, and David Aspinall has written a simple emacs interface for Isabelle[1]. Graphical interfaces to both HOL and Isabelle have been developed by Laurent Théry et al using the Centaur programming environment generator [13, 14].

In our work, we have been motivated by the need to provide support for two basic kinds of user. The first of these is the *verifier*, a verification expert who has detailed knowledge of the capabilities of a given theorem prover, and can use them to advantage to solve a particular problem. The other kind of user is the *engineer*, who is an expert in a particular application domain, but who does not have theorem proving expertise. The verifier needs powerful and flexible theorem proving facilities and will look to a GUI to provide greater productivity; the engineer is looking, above all, for a tool which is easy to use, operates according to a familiar paradigm and in which there is large degree of automation in carrying out proofs. Because of these requirements, it will often be the case that a verifier constructs the special-purpose theorem proving environment suitable for use by an engineer.

The aim of this paper is to describe the ideas behind XIsabelle[2], a Graphical User Interface for Isabelle, which uses the script language Tcl and the Tk Toolkit [16]. XIsabelle communicates with Isabelle by means of the program Expect [17], which is also written in Tcl/Tk[3].

[1]Both these interfaces are available by anonymous ftp from ftp.cl.cam.ac.uk

[2]XIsabelle is available by anonymous ftp from ftp.cl.cam.ac.uk (where Isabelle itself can also be found).

[3]XIsabelle requires Tcl Version 7.4 and Tk Version 4.0, available by anonymous ftp from ftp.smli.com in the directory pub/tcl, and Expect Version 5.18, available by anonymous ftp from ftp.cme.nist.gov in the directory pub/expect.

The use of Tcl/Tk and Expect to design a GUI for a non-graphical interactive program is an extremely simple but powerful technique which has been promoted by Don Libes, the author of Expect [18]. Such GUIs, called X-wrappers, are easy to write and modify, and new features can be added as needed. No changes need to be made to the non-graphical program in order to get the GUI working — indeed, the source code may not always be available.

The implementation of XIsabelle described in this paper is a simple 'front-end' which is easy to understand and modify, if so desired. It works under either Standard ML of New Jersey or Poly/ML, and should be portable with minimal modifications to other ML systems running Isabelle. It does not require any changes to the theorem prover source code. Because Tcl/Tk and Expect are public domain, the use of XIsabelle neither increases the cost nor limits the range of platforms on which Isabelle runs. XIsabelle is usable in conjunction with other tools, in particular with other Tcl/Tk programs.

XIsabelle is more than just a plain graphical 'front-end' to Isabelle. It provides a number of features not directly accessible in Isabelle. Here are the main features:

- It permits theory browsing, i.e. inspecting the structure of a given logical theory.
- It largely removes the need for the user to learn Isabelle commands.
- It allows the user to carry out a proof by 'clicking'.
- It provides advice to the user on possible proof steps.
- Proofs can be saved and replayed at a later stage.

We believe that the present work will benefit the Isabelle community, and that XIsabelle offers a most promising way ahead for an Isabelle GUI. XIsabelle would be a useful tool as a 'gentle' introduction to Isabelle for beginners. Our aim is to stimulate discussion on the user-interface aspects of what is already a very powerful theorem-proving environment.

This paper is organised as follows. In Section 2 we summarise the way in which users wish to interact with proof assistants. Section 3 gives an overview of the Isabelle theorem prover. We then discuss the design philsophy for XIsabelle in Section 4, followed by an overview of the browsing and editing features of XIsabelle (Section 5) and the theorem proving features (Section 6). In Section 7 we work through an example of an interactive proof. Section 8 presents some conclusions.

2 Interaction with Proof Assistants

Proof assistants have a basic notion of *theory*, namely a logical framework in which proofs are carried out. The theory hierarchy is a tree structure: each theory has a *parent* theory, and there is a natural notion of a theory's *ancestors.* A given theorem prover may provide a single fixed but highly flexible framework (as with the HOL system), or else provide a range of given theories from which the user may choose from, along with support for constructing new theories (as with the Isabelle prover).

There are three broad ways (or modes) that the user will interact with a proof assistant. These are *theory browsing*, *theory editing* and *theorem proving*, and are described in detail below.

2.1 Theory Browsing

Theory browsing refers to the exploration of a given theory hierarchy. It can involve one or more of the following activities:

- selecting the theory (or logic) in which to work;
- inspecting the basic constants and axioms of a theory;
- finding out the parent and ancestors of a theory;
- checking to see if a given term is well-formed in a theory; and
- inspecting the theorems of the theory.

2.2 Theory Editing

Theory editing refers to the construction of new theories, or extensions of existing ones. It can involve:

- setting up a new theory;
- editing an existing theory (by altering or extending the syntax, adding new axioms etc.);
- saving a theory; and
- loading a theory into the proof assistant.

2.3 Theorem Proving

Theorem proving activities are the most challenging and difficult aspects of a proof assistant. They include tasks such as:

- setting up a goal to prove;
- inspecting the current proof state;
- choosing a proof step to apply;
- designing a new proof procedure;
- applying a proof step;
- undoing a proof step;
- storing a proved theorem (or a proof); and
- replaying a proof.

Clearly, there is not a sharp division between these three modes of interaction. For example, the user may interrupt a proof in order to carry out some further theory browsing to discover what proof step to apply next.

A key requirement for a proof assistant, and especially for its GUI, is to provide support for the user in carrying out the above tasks, including the ability to change between the different 'modes' of interaction.

3 Isabelle

3.1 Introduction

Isabelle has been under development by Larry Paulson and collaborators at the University of Cambridge since 1986 [19, 7, 20, 21]. Much of the recent work has been carried out by Tobias Nipkow and others at the Technische Universität in Munich. Isabelle is a member of the LCF family [22] of tactical theorem provers, and is written in Standard ML.

Isabelle is a generic prover: the logic of discourse (*object logic*) may be defined by the user, or chosen from one of the object logics provided with the system. Isabelle also has an expressive *meta-logic*, in which the inference rules and axioms of new object logics can be formulated. Isabelle allows concrete syntax, and supports derived forms via user-provided parse and print translations.

Isabelle supports forwards proof, where new theorems are constructed out of existing ones by means of the application of inference rules. It also supports backwards (goal-directed) proof using *tactics*, where a given goal is reduced to subgoals by means of the application of inference rules backwards. Isabelle has a subgoal package, for manipulating proofs interactively. Isabelle lacks the powerful proof heuristics found in systems such as EVES [11], but it is possible to write powerful proof procedures using the tactic language. New tactics can be constructed from existing ones by means of *tacticals*. Various search tactics can be constructed. Answer extraction during proofs is made possible by Isabelle's *scheme variables*.

In this section we shall briefly describe the various aspects of Isabelle (meta-logic, object logics and their theory files, forwards and backwards proof, tactics and tacticals).

3.2 The Meta-Logic

The meta-logic is the framework in which new theories are constructed. It must be compact, but expressive enough to be able to formulate the rules and axioms of the theory. Isabelle's meta-logic is intuitionistic higher-order logic with universal quantification and equality, and a type system with order-sorted polymorphism. The meta-logic is implemented as the theory 'Pure'.

3.3 Object Logics and Theories

An object logic is an ML object of type `theory`. The axioms and rules are of type `thm`. Isabelle comes provided with a number of object logics, including First Order Logic (FOL), Zermelo-Fraenkel Set Theory (ZF) and a version of Higher-Order Logic (HOL). These last two logics are very rich, providing extensive support for the mathematical modelling of systems. The usefulness of ZF and HOL in this regard has been further enhanced by the recently developed inductive definitions package [23].

Isabelle theories are contained in *theory files.* These are of the form 'name.thy', and declare the constants, definitions, rules, syntax translations etc.

Isabelle works with inference rules expressed in a natural deduction style. Each logical connective has, in general, elimination and introduction rules of inference. Here are some examples of rules in first-order logic:

$$\frac{P \quad Q}{P \wedge Q} \quad \text{(conjI)}$$

$$\frac{[P] \; Q}{P \supset Q} \quad \text{(impI)}$$

$$\frac{P \supset Q \quad P \quad [Q] \; R}{R} \quad \text{(impE)}$$

In these rules, the notation $[P]Q$ is a shorthand for $\frac{P}{Q}$. The rule `impE` may look strange: it is a variant of the *modus ponens* rule

$$\frac{P \supset Q \quad P}{Q} \quad \text{(mp)}$$

which is more suitable for elimination.

3.4 Forwards Proof

Isabelle allows new theorems to be created by *resolution.* For example, consider the two theorems

$$\frac{P \wedge Q}{P} \quad \text{(conjunct1)}$$

$$\frac{P}{P \vee Q} \quad \text{(disjI1)}$$

Here the conclusion of `conjunct1` unifies with the assumption of `disjI1`. The resolution of these two is the theorem

$$\frac{P \wedge Q}{P \vee Q} \quad \text{(conjunct1 RS disjI1)}$$

Another, very powerful, method for reasoning forwards is the function `rule_by_tactic`, which takes a given theorem and tactic, and applies the tactic to the theorem regarded as a kind of proof state, to yield a new theorem.

3.5 The Subgoal Package

Isabelle also carries out goal-directed proofs, and contains a subgoal package to assist with interactive proof. A *proof state* consists of a *goal*, along with a number of subgoals whose validity establishes that of the goal. The subgoals can be thought of as proof obligations. Diagrammatically we display a proof state as follows:

$$\frac{initialgoal}{subgoal_1 \ldots subgoal_n}$$

When we set a goal in Isabelle we have as our initial proof state

$$\frac{goal}{goal}$$

in which there is a single subgoal identical with the original goal. A proof state with no subgoals is a proof of the original goal.

3.6 Tactics and Tacticals

Proof states are transformed to new states by means of the application of *tactics*. In Isabelle a tactic may fail, or return one or more new proof states, possibly a lazy infinite list.

In general, if T is a tactic and ϕ is a proof state, then the result $T\phi$ of applying T to ϕ is written as a list to capture the various alternatives:

$$T\phi = [\,] \quad (failure)$$

$$T\phi = [\psi] \quad (unique\ result)$$

$$T\phi = [\psi_1, \psi_2, \psi_3, \ldots] \quad (multiple\ outcomes)$$

If the tactic succeeds, the head of this list is the active proof state, and is usually presented with all of its subgoals shown. Many tactics act on a number of subgoals, automatically instantiating variables and renumbering the subgoals as appropriate.

Pure Isabelle has a number of commonly used basic tactics (object logics also have their own special purpose tactics). We shall discuss the most important of these.

Recall that Isabelle emphasises the natural style of reasoning; correspondingly, most proof steps are carried out by means of backwards reasoning using inference rules of the logic. This is also called *resolution*. Isabelle provides a single ML function to do this, called in this way: `resolve_tac thms i`. This tactic tries each theorem in the list `thms` against subgoal `i` of the proof state, until a rule is found whose conclusion can unify with the subgoal.

Other tactics are `assume_tac i`, which tries to solve subgoal `i` by assumption; (meta-level) rewriting tactics such as `rewrite_goals_tac thms`, which uses the given definitional theorems to expand the proof state; and (object-level) rewriting such as `simp_tac ss i`, which simplifies subgoal `i` using a given set `ss` of object-level rules.

Tactics may be combined together using *tacticals*, the most important of which are given in Table 1.

Table 1: *Tacticals*

Tactical	Description
`tac1 THEN tac2`	sequencing
`tac1 ORELSE tac2`	choice
`REPEAT tac`	iteration
`DEPTH_FIRST pred tac`	search

3.7 Example Proof

To illustrate interactive proof in Isabelle, consider the following result in First Order Logic:

$$a \wedge b \supset b \wedge a.$$

We set up the goal with the command

```
> goal FOL.thy "a & b --> b & a";
Level 0
a & b --> b & a
 1. a & b --> b & a
val it = [] : thm list
```

Here is a complete step-by-step proof, showing Isabelle's responses.

```
> by (resolve_tac [impI] 1);
Level 1
a & b --> b & a
 1. a & b ==> b & a

 > by (eresolve_tac [conjE] 1);
Level 2
a & b --> b & a
 1. [| a; b |] ==> b & a
```

```
> by (resolve_tac [conjI] 1);
Level 3
a & b --> b & a
 1. [| a; b |] ==> b
 2. [| a; b |] ==> a

> by (assume_tac 1);
Level 4
a & b --> b & a
 1. [| a; b |] ==> a

> by (assume_tac 1);
Level 5
a & b --> b & a
No subgoals!
```

The first proof step involves recognising that $\supset$ is the outermost connective, and that we need to apply resolution with its introduction rule. We also need to know the ML name (in this case, **impI**) of this rule. This knowledge does come from experience, reading the manuals etc., but it can be time-consuming for a beginner to know what to do. In this case, the rule **impI** is contained in the theory file IFOL.thy. The command **axioms_of IFOL.thy** can be used to confirm this, or the source code consulted.

The second and third proof steps involve resolution with derived rules (these are rules which are not part of the original theory, but proved subsequently). In many cases (such as the logics FOL, LK, ZF and HOL), such rules are now stored in a database by Isabelle[4]. The command **thms_of IFOL.thy** lists these rules.

The final two proof steps solve the goal by making use of assumptions.

It can be seen that quite a deal of knowledge is required of the user in order to carry out proofs in Isabelle. For our simple proof, Isabelle provides a general-purpose tactic called **fast_tac** which proves this goal in one step, i.e.

```
by (fast_tac FOL_cs 1);
```

However, in general, we need to be able to carry out interactive proofs. We would like a proof assistant to help us decide which proof step to apply next, as well as a simple means for inspecting available rules and theorems. We would also like the ability to edit theories and ML files, and to save and load partial proofs.

The aim of the XIsabelle tool is to provide such assistance by means of a graphical user interface to Isabelle, using simple mouse-based commands. The following sections describe XIsabelle.

[4]XIsabelle can only be used for those logics whose derived theorems are stored in a database

4 XIsabelle: Design and Implementation

XIsabelle has been designed to provide an easy-to-use graphical interface to Isabelle.

XIsabelle is written entirely using Tcl/Tk and Expect commands, and contains no C code. A small amount of ML code (in Tools.ML) is needed to provide the necessary interface to Isabelle, but no modifications to the Isabelle source code are needed.

4.1 Overview of Tcl/Tk and Expect

Tcl is a string-based interpreted shell language, developed by John Ousterhout and described in detail in his book [16]. It provides support for lists, and simple interaction with Unix. When combined with the Tk widget set, X-windows applications can be created simply and easily. The program `wish` allows Tcl/Tk commands to be entered interactively. As an example, we start up `wish`, and enter the commands

```
button .b -text "Push" -command {exit}
pack .b
```

The `wish` program puts up a top-level widget (called '`.`'). The `button` command creates a button widget called '`.b`' which contains the text '`Push`' and is bound to the command `exit`. The button is not visible until it is packed within the top level widget with the second command. If we click with the mouse on this button, the effect is to exit the `wish` program.

The Expect program [17] is one of the best-known applications written in Tcl. Expect was developed by Don Libes at the National Institute of Standards and Technology. It provides a powerful means of communication with other processes. Once a process has been spawned by means of the command `spawn`, the command `exp_send` can be used to send information to the process. The command `expect` can match the possible resulting patterns from the process, taking appropriate action.

The program Expectk combines all the features of Tcl/Tk and Expect, and makes possible the construction of graphical interfaces, or 'X-wrappers', to non graphical programs. XIsabelle has been built using Expectk.

4.2 Interaction with Isabelle

In order to understand the basic means by which XIsabelle interacts with Isabelle, we need to review the way commands are entered in ML. In general, if `dec` is a declaration, then entering the command

```
dec;
```

at the keyboard will make ML respond with the *value* and *type* of the declaration. In some cases `dec;` will in addition have the side-effect of printing something, called the *output*. For example, suppose that `dec` is the string `val x = 3`. The result of entering

```
val x = 3;
```

in ML is

```
val x = 3 : int
```

On the other hand, consider the ML expression `prs "hello\n"`, which is automatically regarded by ML as the declaration `val it = prs "hello\n"`. This time, the result is

```
hello
val it = () : unit
```

in which the string `"hello\n"` is the output.

The means by which XIsabelle interacts with Isabelle is summarised in Figure 1.

The Tcl procedure `MLdec` uses the basic Expect commands `exp_send`, to send the request to Isabelle, and `expect`, to respond to the result of executing the ML code. Thus, the Tcl command

```
MLdec dec
```

has the same effect as if

```
dec;
```

had been entered in ML. The procedure `MLdec` returns the Tcl list `[output,value,type]` for subsequent use by XIsabelle.

It can take a significant amount of time for a declaration to be sent off by XIsabelle and a result returned by ML. For example, the selection of the ZF theory within the ZF logic means that a very large list of theorems has to be displayed. Ideally, other mouse operations within the XIsabelle window should not be allowed while this is going on[5].

[5] At this stage, our implementation does not prevent this.

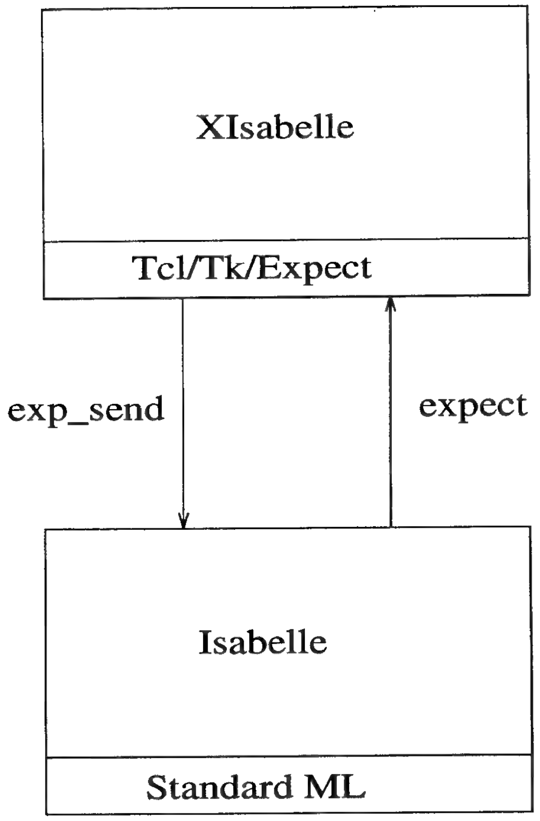


Figure 1: *XIsabelle: Interaction with Isabelle*

4.3 ML or Tcl?

It is often the case that a particular data structure or function can be implemented either in ML or in Tcl, and we need to make a decision between what are two quite different kinds of language. All things being equal, we would prefer ML over Tcl, because it is easier to write correct code. In practice, a clear choice is usually indicated: we write ML code to set up and manipulate data structures and functions closely linked to Isabelle, and Tcl/Tk code for the data structures and operations affecting XIsabelle widgets.

For example, in XIsabelle we need to have access to the ancestors of a theory. There is already an ML function `parents_of`, so the most sensible thing is to use it to write a new ML function called `ancestors_of`. On the other hand, the code which updates the scrolled list of theory ancestors in XIsabelle has to be written in Tcl/Tk.

In some cases, the choice is not so clear — for example, should the theories in a given logic be represented as an ML list or as a Tcl list? In this case, we chose Tcl because most of the operations relate to the scrolled list in the XIsabelle Browser.

5 Theory Browsing and Editing

We now describe in detail how XIsabelle works, beginning with theory browsing and editing. It is assumed that Isabelle has already been built using either Standard ML of New Jersey or Poly/ML, that the program Expectk is available on the system (Expectk is built using Tcl/Tk and Expect), and that the source code for XIsabelle has been installed according to the instructions. XIsabelle is invoked as follows. At the Unix prompt, enter

```
xisabelle [-l Logic] [-i Interp]
```

from within an xterm or other shell window. There are two optional arguments. The first of these is the Isabelle object-logic, which must be one of FOL, ZF, LK or HOL (the default is FOL). This object logic remains fixed throughout a session with XIsabelle. The second argument is the name of the Tcl program that called XIsabelle (if any). This is provided so that XIsabelle can be used as part of a suite of Tcl programs: each of these programs communicates with the others via the Tcl `send` command.

If XIsabelle has started successfully, the theory browsing tool (called the Browser) will appear, shown in Figure 2. Note that the original xterm or shell from which XIsabelle was started allows direct interaction with Isabelle at any time if necessary.

5.1 Browsing

The menu bar at the top of the screen has three menu buttons: File, Term and Help. The operations available under each of these are summarised in Table 2, where the corresponding accelerator keys are also shown [6].

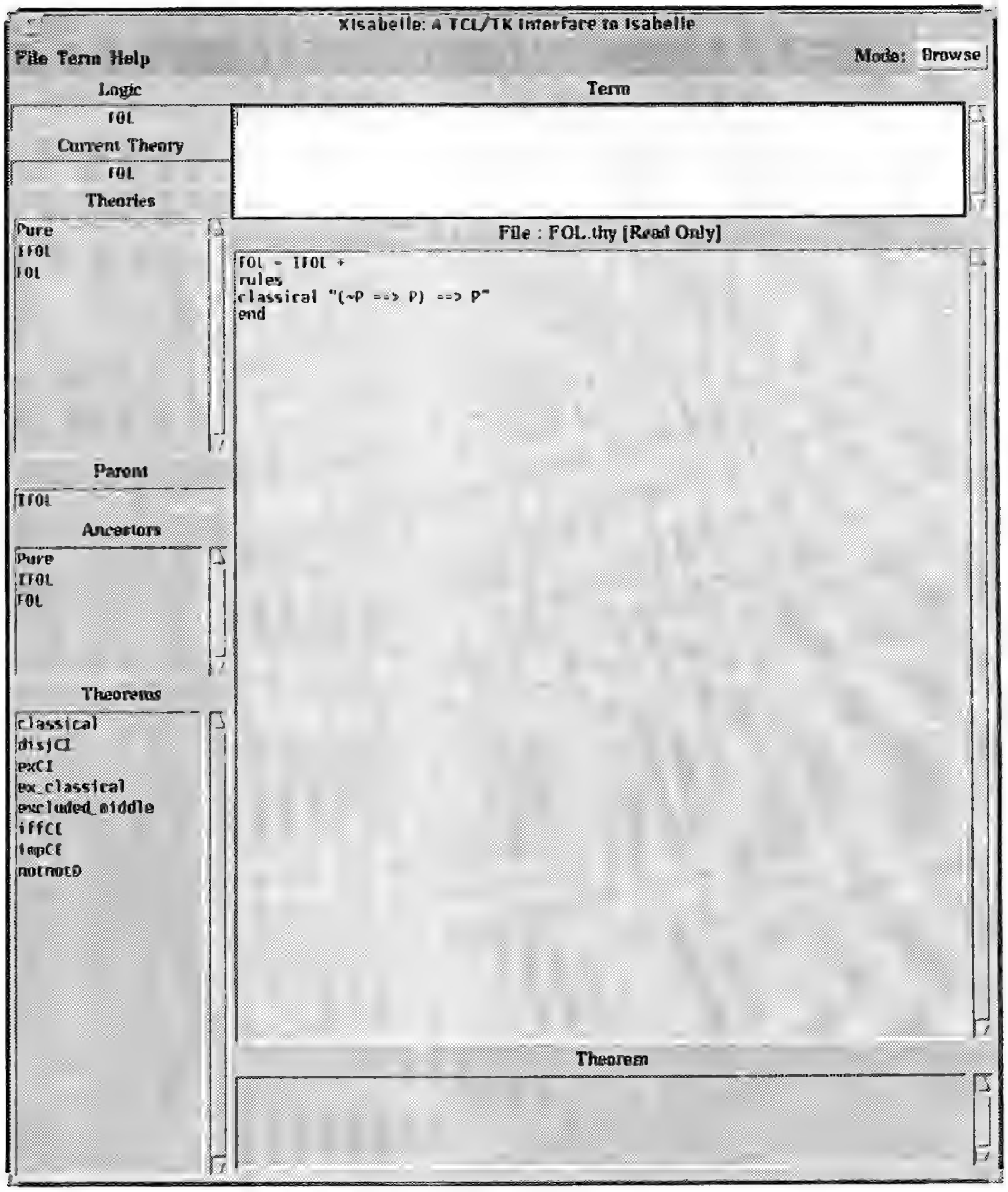
On the right of the Browser is a mode button, which toggles between Browse and Edit Mode. At the top left of the Browser are small windows showing the current logic (FOL), and the current theory (also FOL).

We shall introduce the other parts of the Browser by discussing in turn the various theory browsing activities described in Section 2.

5.1.1 Selecting a Theory

Below the current theory window is the Theories window, showing all the theories present in this logic. In this case there are only three: FOL itself, Intuitionistic First Order Logic

[6]In this and subsequent tables, the accelerator keys for each function will also be shown. These are of the form Meta-key, where Meta is the diamond-shaped key on Sun keyboards.



```
FOL = IFOL +
rules
classical "(~P ==> P) ==> P"
end
```

Figure 2: *XIsabelle at Startup*

Table 2: *Browser Operations*

Menu	Operation	Description	Accelerator
File	New	Create a New Theory File for Editing	Meta-N
	Open	Open an Existing Theory File for Editing	Meta-O
	Save	Save the Theory File	Meta-S
	Use	Load a Theory or ML File into Isabelle	Meta-U
	Load Proof	Load a Proof into XIsabelle	Meta-F
	Quit	Quit XIsabelle	Meta-Q
Term	Check	Check the Syntax of a Given Term	Meta-C
	Prove	Invoke the Prover	Meta-P
Help	Help	Display Help Information	Meta-H
	Documentation	Preview XIsabelle Documentation	Meta-D

(IFOL), and Pure Isabelle (the meta-logic). Browsing is achieved by clicking with the left mouse button on one of the theories in the list. For example, Figure 3 shows the effect of clicking on the theory IFOL. The parent and ancestors change to reflect the current theory. The current theory window will also change to IFOL.

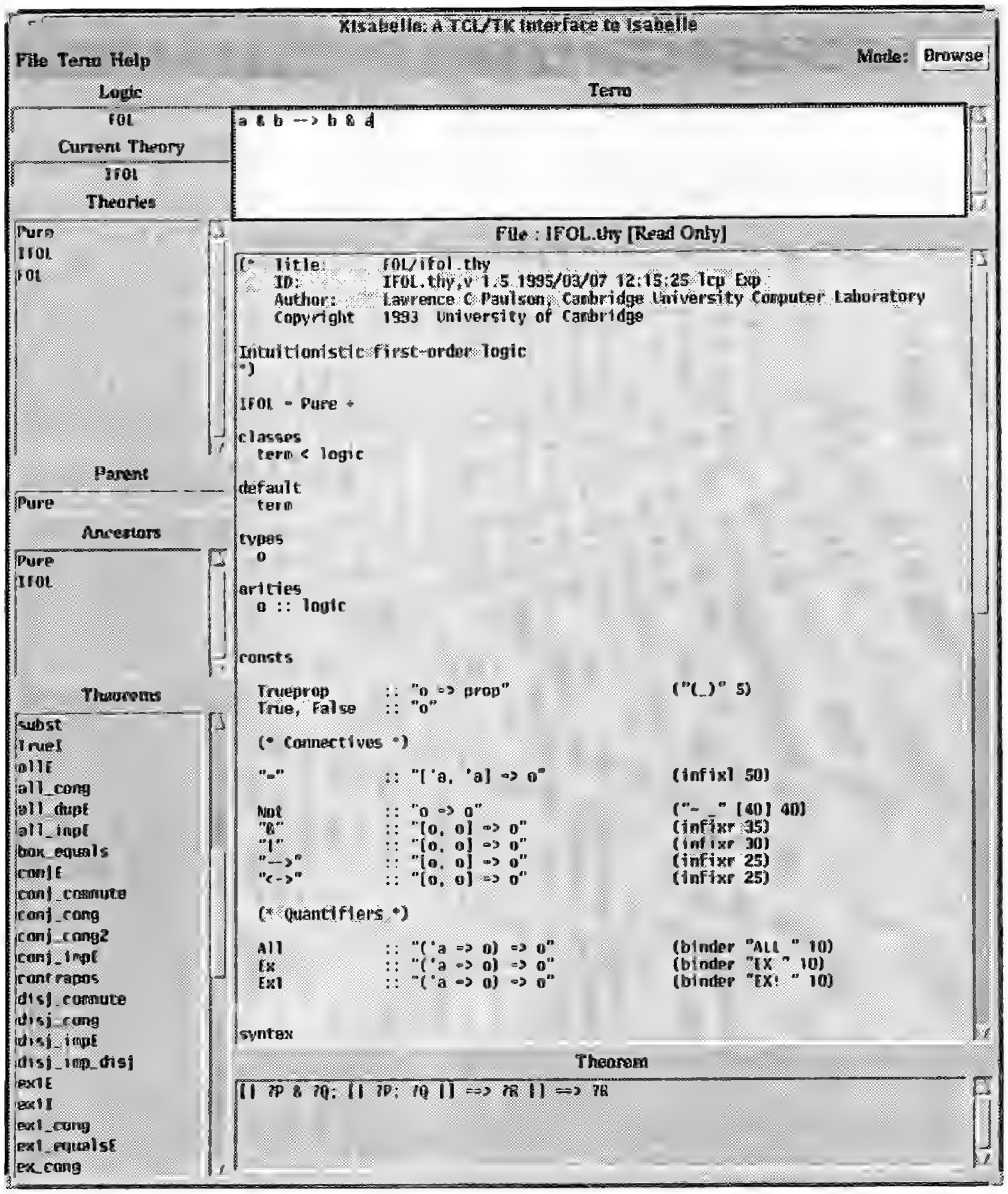
5.1.2 Inspecting a Theory

The large window on the right is the file window. It contains the Isabelle source code for the relevant theory file for the current theory. It cannot be edited at this stage, but can be scrolled. It also changes as different theories are selected[7].

5.1.3 Parent and Ancestors of a Theory

This has already been covered: the action of selecting a theory shows the parent and ancestors of that theory. For the case of FOL, Figure 2 shows the parent (in this case IFOL), and the ancestors (FOL, IFOL and Pure).

[7]It was a deliberate design decision to display a theory's structure by means of showing the theory file. Other tools, notably Mural, have various windows showing the constants, axioms etc of a theory. We could have done this, but we believe that it makes a tool more cumbersome to use. The theory file shows the same information, and is reasonably easy to understand. As a point of principle, if static information about an object such as a theory is to be displayed to the user, then as much information as is sensible and likely to be useful should be presented at one time



Theories:

```
Pure
IFOL
FOL
```

Parent:

```
Pure
```

Ancestors:

```
Pure
IFOL
```

Theorems:

```
subst
TrueI
allE
all_cong
all_dupE
all_impE
box_equals
conjE
conj_commute
conj_cong
conj_cong2
conj_impE
contrapos
disj_commute
disj_cong
disj_impE
disj_imp_disj
ex1E
ex1I
ex1_cong
ex1_equalsE
ex_cong
```

Term:

```
a & b --> b & a
```

File : IFOL.thy [Read Only]

```
(*  Title:       FOL/ifol.thy
    ID:          IFOL.thy,v 1.5 1995/03/07 12:15:25 lcp Exp
    Author:      Lawrence C Paulson, Cambridge University Computer Laboratory
    Copyright    1993  University of Cambridge

Intuitionistic first-order logic
*)

IFOL = Pure +

classes
  term < logic

default
  term

types
  o

arities
  o :: logic


consts

  Trueprop      :: "o => prop"                   ("(_)" 5)
  True, False   :: "o"

  (* Connectives *)

  "="           :: "['a, 'a] => o"               (infixl 50)

  Not           :: "o => o"                      ("~ _" [40] 40)
  "&"           :: "[o, o] => o"                 (infixr 35)
  "|"           :: "[o, o] => o"                 (infixr 30)
  "-->"         :: "[o, o] => o"                 (infixr 25)
  "<->"         :: "[o, o] => o"                 (infixr 25)

  (* Quantifiers *)

  All           :: "('a => o) => o"              (binder "ALL " 10)
  Ex            :: "('a => o) => o"              (binder "EX " 10)
  Ex1           :: "('a => o) => o"              (binder "EX! " 10)


syntax
```

Theorem:

```
[| ?P & ?Q; [| ?P; ?Q |] ==> ?R |] ==> ?R
```

Figure 3: *Browsing IFOL*

5.1.4 Checking a Term

The term window at the top-right is a text widget allowing the entry of terms for syntax-checking, and also goals to give to the Prover (see later).

The Check button (found in the Term Menu) provides a way of checking that the term in the term window is syntactically well-formed for the current theory.

For example, if the current theory is First Order Logic (FOL), then the following term is well-formed:

```
P --> P | Q
```

Pressing the Check button will display the notice

```
Syntax Correct.
```

However, the following term is not well-formed:

```
P --> P Q |
```

Pressing the Check button will display the notice

```
Syntax Error at "Q |"
```

Here is another example. The term below uses the unknown symbol "@", and is thus not well-formed:

```
P --> P | Q@
```

Pressing the Check button will display the notice

```
Lexical Error at "@"
```

The Prove button invokes the Prover, and will be covered in the next Section.

5.1.5 Inspecting Theorems

At the bottom-left are shown the names of all the currently available theorems for the current theory: these consist of the axioms along with those derived theorems stored away in the theory's database by Isabelle. Clicking on a theorem will display the theorem itself in the Theorem window (bottom-right window). This makes use of the Isabelle function `thms_of`.

5.2 Editing

XIsabelle has two modes of use, called Browse mode and Edit mode. For normal browsing operations, XIsabelle is in Browse mode. XIsabelle also supports the editing of user-defined theories and ML files in the file window. Editing is only permitted when XIsabelle is in Edit mode. The Edit/Browse button shows the current mode; clicking on this button with the left mouse button toggles between Browse and Edit modes.

When we are editing a file, one can think of XIsabelle as providing the typical file operations one would expect of a word processor, namely 'New', 'Open' and 'Save'. In addition to these basic operations, we have the Isabelle-specific operations 'Use' and 'Load Proof'. These are all found on the File menu (see Table 2). The 'Load Proof' function will be discussed in the next section.

5.2.1 Setting Up a New Theory

The New button allows the user to construct a new theory or ML file.

When New is selected, the user will be prompted for the name of the file. XIsabelle requires that the file name be a theory file, of the form 'name.thy', or an ML file of the form 'name.ML'. (This is consistent with the Isabelle convention for filename extensions). If the file is a theory file, the Current Theory window will display its name.

XIsabelle is put into Edit mode, and a simple template is provided for editing this new file in the file window.

5.2.2 Editing an Existing Theory

The Open button allows the user to open a theory or ML file for editing.

When Open is selected, a directory navigator will appear, with directories given on the left-hand side and theory and ML files on the right. Note that theory files must be of the form 'name.thy', and ML files of the form 'name.ML'.

The user can navigate between directories by selecting and double-clicking with the left mouse button. Double-clicking on a file, or clicking on the Open button, will open the selected file for editing.

XIsabelle will go into Edit mode, and the file will appear in the XIsabelle's file window.

5.2.3 Saving a Theory

The Save button allows the user to save an Isabelle theory or ML file. This operation must be confirmed.

5.2.4 Loading a Theory into Isabelle

The Use button allows the user to load an Isabelle theory or ML file.

If XIsabelle is in Edit mode, then an attempt will be made to load the theory or ML file currently being edited (to be precise, the Isabelle command `use_thy name` will be carried out in the case of a theory, and the command `use name.ML` will be carried out for a theory file).

If XIsabelle is in Browse mode and the Use button is selected, then a directory navigator will appear, with directories given on the left-hand side and theory and ML files on the right. Navigation is achieved exactly as for opening files (above).

Any errors found while the theory or ML file is being loaded will be reported. If all goes well, the file will appear in the theory file window of XIsabelle. A new theory will be shown on the scrolled list of Theories.

Note: if the Edit/Browse button is selected at any time during editing, XIsabelle will disable edits, go into Browse mode, and attempt to load the file in the file window.

5.3 Getting Help

To get general help, click on the Help button (Help Menu). To get help on a specific menu button, or on the mode button, the left shift key can be held down while clicking on that button.

5.4 Quitting XIsabelle

The Quit button allows the user to quit XIsabelle. If a file is being edited, and has not been saved, then the user is asked to confirm. Otherwise, XIsabelle will silently exit.

6 The Prover

The Prover is a separate top-level window which provides a GUI for Isabelle's subgoal package for interactive proof. An example is shown in Figure 5. This screen snapshot shows the Prover's appearance for the First Order Logic goal discussed in Section 3:

$$a \wedge b \supset b \wedge a$$

In the next section, we shall work through this example in detail. For the present, we provide an overview of the Prover.

The operations available on the Prover's menus are summarised in Table 3.

Table 3: *Prover Operations*

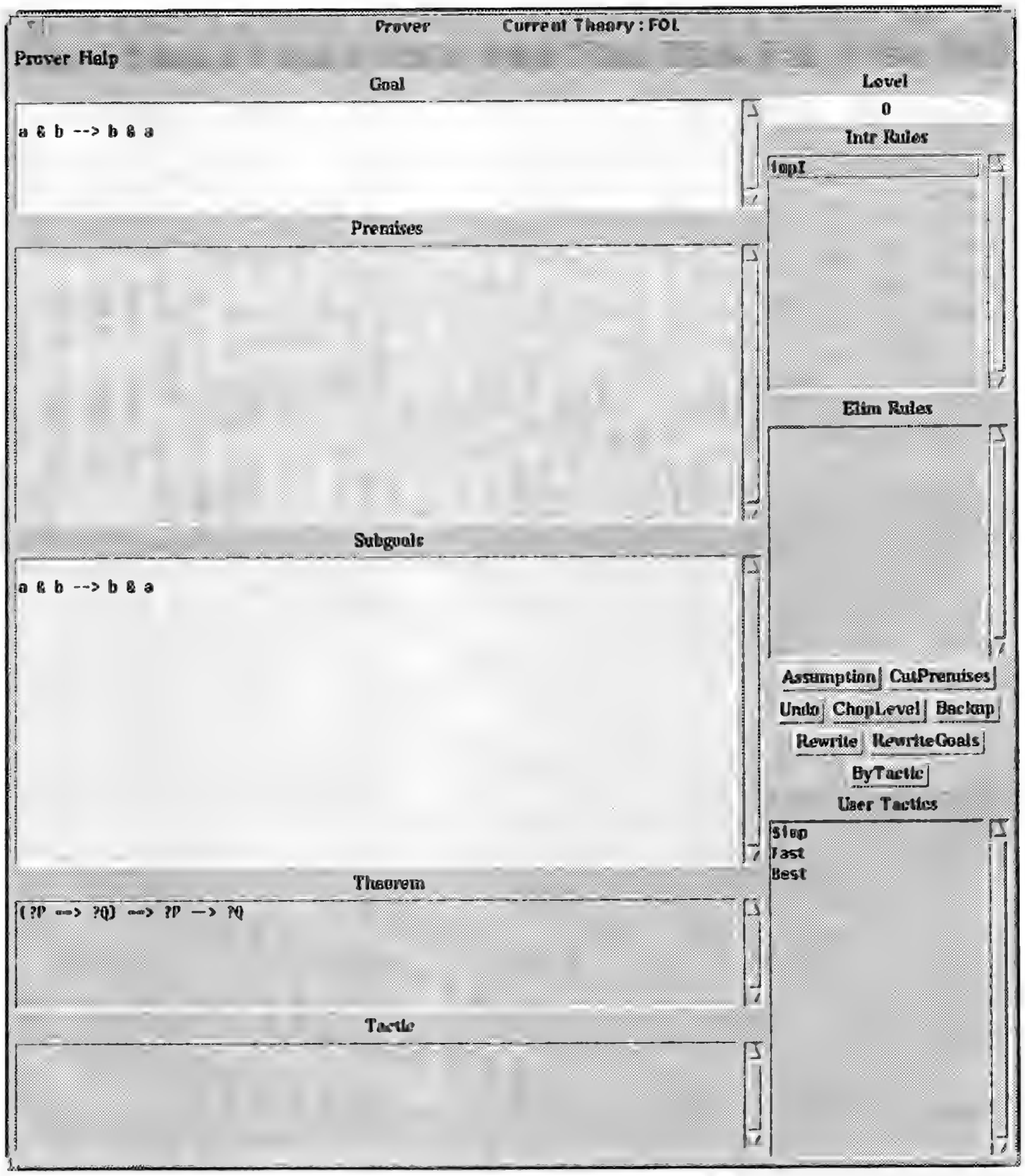
Menu	Operation	Description	Accelerator
Prover	InstallTactic	Install a User-Defined Tactic	Meta-I
	EditTactic	Edit a User-Defined Tactic	Meta-E
	RemoveTactic	Remove a User-Defined Tactic	Meta-R
	ShowTactic	Display Cumulative Proof Tactic	Meta-T
	ProofHistory	Display Proof Steps Carried Out	Meta-Y
	LoadProof	Load a Proof into XIsabelle	Meta-F
	SaveProof	Save a Proof	Meta-V
	AddTheorem	Add a Theorem to the Theory Database	Meta-A
	QuitProver	Quit the Prover	Meta-Q
Help	Help	Display Help Information	Meta-H

6.1 Selecting a Theory

Below the current theory window is the Theories window, showing all the theories present in this logic. In this case there are only three: FOL itself, Intuitionistic First Order Logic (IFOL), and Pure Isabelle (the meta-logic). Browsing is achieved by clicking with the left mouse button

6.2 Setting up a Goal to Prove

The link between the Browser and the Prover is the Prove button on the Term Menu. This button invokes the Prover, provided there is a syntactically correct formula in the term window.



```
a & b --> b & a
```

```
a & b --> b & a
```

```
(?P --> ?Q) --> ?P --> ?Q
```

Figure 4: *XIsabelle: The Prover*

Note: it is possible to continue to use XIsabelle to browse theories even if the Prover is active. However, the theory used in the Prover remains fixed: it is the theory that the Browser was using at the time the Prover was invoked. This theory is displayed in the Prover's theory window.

6.3 Inspecting the Current Proof State

The current proof state is displayed as follows. The Goal window shows the goal (it may have had some scheme variables instantiated during the proof). The Premises window shows the premises, or assumptions, which can be used to prove the goal. These can be regarded as theorems. One or more of these can be highlighted, and introduced into the goal by means of the CutPremises button (see below). Finally, the current subgoals are shown in the Subgoals window. Only one of these may be highlighted at a time: certain tactics are applied to the highlighted subgoal (see below).

Selecting the Show Tactic button will display a window showing the tactic which effects the proof so far. This tactic consists exactly of the steps in the proof history scrolled list, joined together by the tactical THEN.

6.4 Choosing a Proof Step to Apply

On the right-hand side of the Prover are scrolled lists which show the names of introduction and elimination rules which match the active subgoal. Selecting the name of a rule will display the rule in the Theorem window. Double-clicking on the rule name will apply the appropriate introduction or elimination rule (see below).

The next section describes the choice of a proof step in detail for an example proof.

6.5 Designing a New Proof Procedure

The Install Tactic button allows the installation of new tactics in the Prover. These can then be used in proofs for the rest of the session. Such tactics are called *temporary.* Alternatively, they can be saved to the user's tactics file (see below). Such tactics are called *permanent.*

Selecting the Install Tactic button will bring up a text window. The tactic must be entered in the form

```
   name=tactic
```

The tactic (if it type-checks correctly). will be installed on the scrolled list of tactics shown on the right of the Prover window.

XIsabelle permits both tactics and tactic functions (of type int → tactic). If a tactic function is applied during the proof, XIsabelle will assume that it is to be applied to the active (highlighted) subgoal.

When the Prover is invoked, XIsabelle looks for a file of the form .logic.tactics (such as .FOL.tactics), first in the XIsabelle directory, then in the user's home directory, and finally in the directory from which XIsabelle was started. These files must contain lines of the form:

```
name=tactic
```

XIsabelle will load them in turn, and install all these (permanent) tactics in the Prover.

The Remove Tactic button allows the removal of user tactics in the Prover. Selecting the Remove Tactic button has the effect (after user confirmation) of removing the selected tactic from the tactics list in the Prover window. If the tactic is temporary (i.e. was created for the current session only), then it can only be removed for the current session. If it is permanent (i.e. was present in the tactics file), then it can be removed permanently. The tactic will no longer be available.

An error occurs if a tactic is not selected.

The Edit Tactic button allows the editing of existing tactics. For a permanent tactic, the tactics file is updated; for a temporary tactic, the changes have effect only in the current session. Note that both the name and the definition of the tactic may be edited.

6.6 Applying a Proof Step

Various simple proof steps can be applied with XIsabelle, as follows. The equivalent Isabelle commands are summarised in Table 4. In this table `i` denotes the active subgoal.

In all cases, if the command is successful, the proof state will change. If the command fails, an error dialog will appear, and the proof state will not be changed.

6.6.1 Introduction and Elimination Rules

These are the basic proof steps, and are carried out by double-clicking on an introduction or elimination rule. The resolution step is performed on the active subgoal.

6.6.2 Assumption

The Assumption button tries to solve the active subgoal by assumption. For this step to succeed, certain scheme variables may need to be instantiated.

6.6.3 CutPremises

The CutPremises button allows the insertion of selected premises into a subgoal, for use in subsequent proof steps.

6.6.4 ChopLevel

The ChopLevel button goes to a specified level in the proof. A response widget will appear, asking for the level to be specified.

6.6.5 Backup

The Backup button backs up to an alternative proof state (if there is one).

6.6.6 Rewrite

The Rewrite button carries out (meta)rewriting of the entire proof state with the given definitions. The definitions to be used for rewriting are given as a comma-separated list in a response widget.

6.6.7 RewriteGoals

The RewriteGoals button is similar to Rewrite, except that only the subgoals are rewritten (not the main goal).

6.6.8 ByTactic

The ByTactic button is used to apply a general tactic or tactic function. A response widget will appear, asking for the tactic or tactic function to be specified. XIsabelle will

Table 4: *Proof Steps*

Proof Step	Isabelle Equivalent
Introduction Rule	`by (resolve_tac[rule] i);`
Elimination Rule	`by (eresolve_tac[rule] i);`
Assumption	`by (assume_tac i);`
CutPremises	`by (cut_facts_tac premises i);`
ChopLevel	`choplev n;`
Backup	`back();`
Rewrite	`rewrite_tac defs;`
RewriteGoals	`rewrite_goals_tac defs;`
ByTactic	`by tac;`
UserTactic	`by usertac;`
Undo	`undo();`

figure out whether this a legal tactic (of type tactic) or a legal tactic function (of type int $\rightarrow$ tactic). An expression of any other type is illegal, and an error dialog will be shown.

If a tactic is to be frequently applied, it should be installed as a new user tactic (see above in 6.5).

6.6.9 UserTactics

On the bottom right hand side of the Prover are found a scrolled list of User Tactics. These tactics are installed by XIsabelle when the Prover starts up. They are defined in the file 'tactics.ini', and can be chosen to suit user preferences. If a user tactic is selected, its definition appears in the Tactic window. Double-clicking the tactic has the effect of applying the tactic to the current proof state.

6.7 Undoing a Proof Step

The Undo button allows a proof step to be cancelled, so that Isabelle reverts to the previous proof state[8]. Note that Undo will fail if the proof state is at level zero[9].

[8]The Undo key (L4) on Sun keyboards has the same effect.

[9]Note that Isabelle sometimes allows the Undo operation in these circumstances. However, XIsabelle does not.

6.8 Storing Theorems and Proofs

6.8.1 Add Theorem

The Add Theorem button adds a theorem to the theory's database of derived theorems, provided that it has just been successfully proved interactively.

6.8.2 Save Proof

The Save Proof button saves the current proof to a file (this proof may be incomplete). When Save Proof is selected, XIsabelle will request the name of the proof (if the proof is complete, this will also be the theorem name). The proof script will be saved in the file name.prf.

6.8.3 Proof History

The Proof History button displays all the steps carried out in the proof so far.

Selecting the Proof History button will pop up a new top-level window, showing a scrolled list of the tactics which have (successfully) been applied so far. Tactics which failed are not shown; on the other hand, tactics which succeeded but did not change the proof state are shown. As the proof proceeds, the list is updated.

The Proof History can be used to go back to an earlier level in the proof. Double-clicking on an tactic in the Proof History will chop the level to the point at which that tactic had just been applied.

An error occurs if a tactic is not selected.

6.9 Replaying a Proof

The Load Proof button allows a proof file to be opened[10]. When Load Proof is selected, a directory navigator will appear, with directories given on the left-hand side and proof scripts on the right. Note that proof scripts must be of the form 'name.prf'.

The user can navigate between directories by selecting and clicking with the left mouse button. Clicking on a theory file, or on the Open button, will load the selected proof.

[10]This button is also available on the Browser File menu.

If the proof file is successfully loaded, the Prover will appear (if one is not already present), and the proof will be executed exactly in the sequence it was originally carried out.

6.9.1 Quit Prover

The Quit Prover button will quit the Prover. When Quit Prover is selected, if the proof is currently at level 0 (the top-level), XIsabelle will assume that the proof does not have to be saved; otherwise, XIsabelle will ask if the user wishes to save the proof to a file. This operation will not quit XIsabelle itself.

7 Example proof

In this section, we shall illustrate the various features of XIsabelle's Prover by working through the example already mentioned in Section 6, namely the following result in First Order Logic.

$$a \wedge b \supset b \wedge a$$

In FOL, enter the goal

```
a & b --> b & a
```

in the Term window. Invoke the Prover by clicking on 'Prove' in the Term Menu. If there is no syntax error, the Prover will appear as shown in Figure 4.

Note that the Prover has its own Theory window (showing FOL in this case). The Level window shows the current level of the proof state (intially level 0).

The Goal window shows the goal exactly as it appeared in the Browser's term Window. (The difference between the two lies in the fact that scheme variables may appear in the goal, and be instantiated later during the proof, but this is not the case here.)

The Premises window is empty because the goal has no premises. The Subgoals window intially shows just one (highlighted) subgoal, which is identical with the original goal.

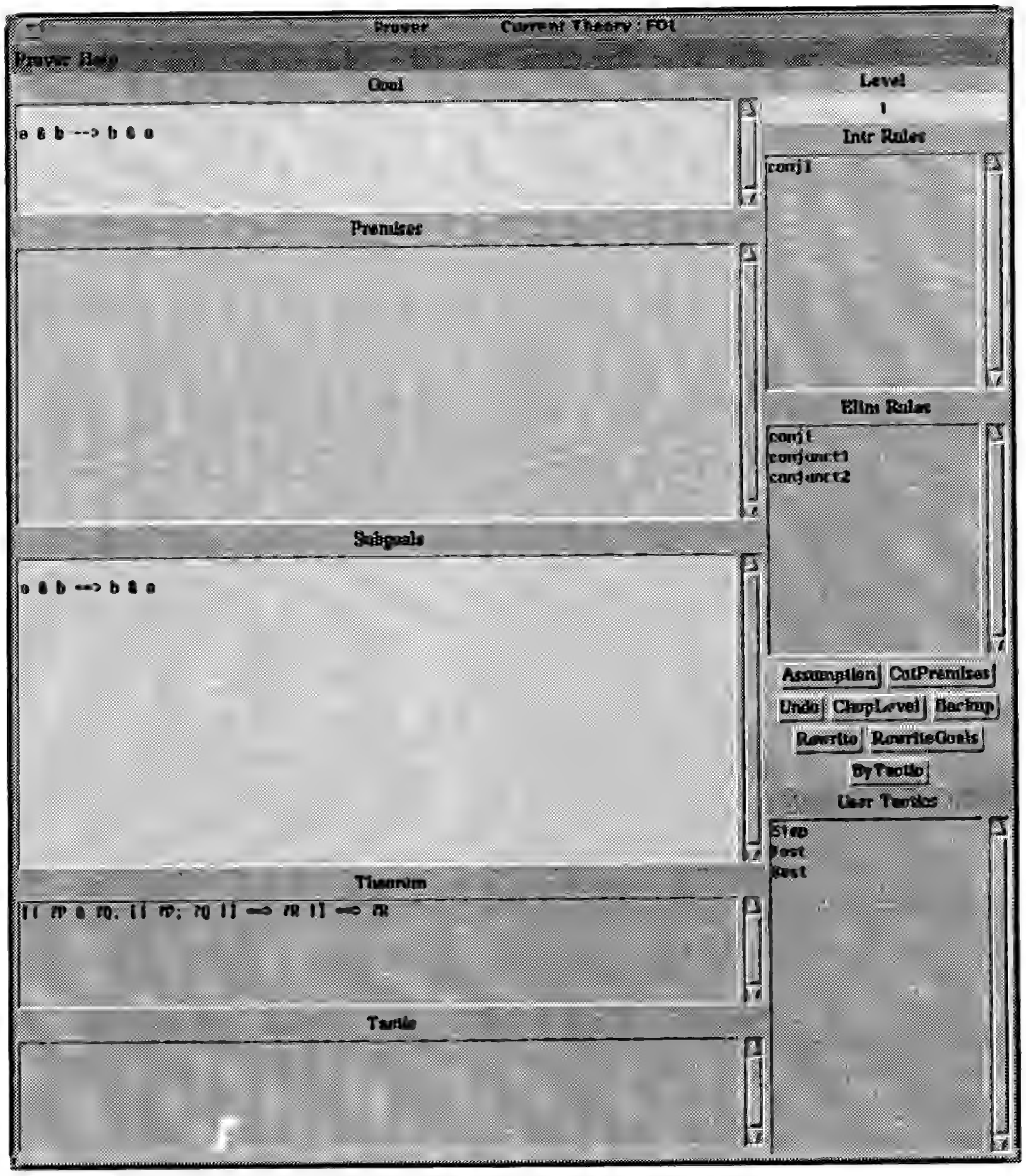
There is only one rule suitable for resolution, and this is the introduction rule

$$\frac{[P] \;\; Q}{P \supset Q} \qquad \text{(impI)}$$

The rule `impI` appears in the scrolled list of introduction rules, and is displayed in the Theorem window.

Now we are in a position to progress the proof. XIsabelle has told us that the rule `impI` is suitable to use for resolution. To apply the rule `impI`, we double-click on it, reaching the stage shown in Figure 5. In this picture, the matching elimination rule `conjE` is shown displayed in the Theorem window. Note that the subgoal has changed to one involving meta-implication, and also that the Level of the proof state has changed to 1 (in the Level window).

XIsabelle is now telling us that the introduction rule `conjI`, and the elimination rules `conjE`, `conjunct1`, `conjunct2` and `rev_mp` can be resolved with the subgoal. These theorems can be inspected in the Theorem window by selecting with the mouse. In order to keep control of the number of subgoals, we decide to apply `conjE` first, reaching Figure 6.



```
a & b --> b & a
```

```
conjI
```

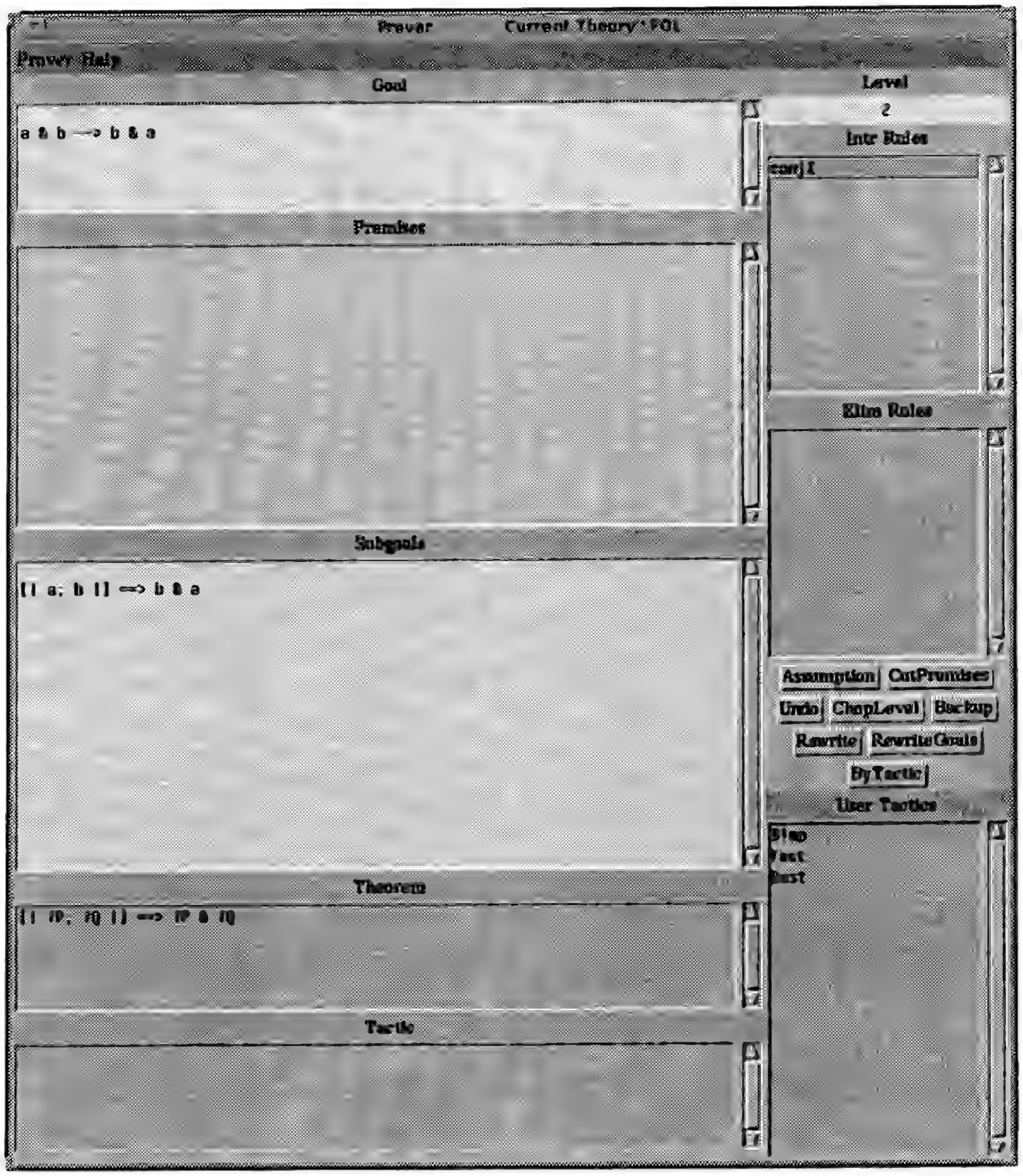
```
conjE
conjunct1
conjunct2
```

```
a & b --> b & a
```

```
Step
Fast
Best
```

```
[| ?P & ?Q; [| ?P; ?Q |] ==> ?R |] ==> ?R
```

Figure 5: *Example Proof (cont)*



```
a & b --> b & a
```

```
[| a; b |] ==> b & a
```

```
[| ?P; ?Q |] ==> ?P & ?Q
```

Figure 6: *Example Proof (cont)*

Now we are at level 2, with still only one subgoal to prove. We apply the rule `conjI`, giving level 3, with two subgoals as shown in Figure 7.

These two subgoals are trivially solved by highlighting them in turn, and clicking on the Assumption button. The final stage of the proof is shown in Figure 8.

The complete proof can be seen by using the ProofHistory command (Prover menu), which shows the tactics applied so far:

The proof can be saved (and a name given to the theorem just proved) by using SaveProof.

Having carried out the proof in single steps, we note that it can also be done in one step. If we revert to the initial proof state (by pressing the ChopLevel button, and choosing level 0), then we can prove the goal by double-clicking on **`fast`** in the list of User Tactics.

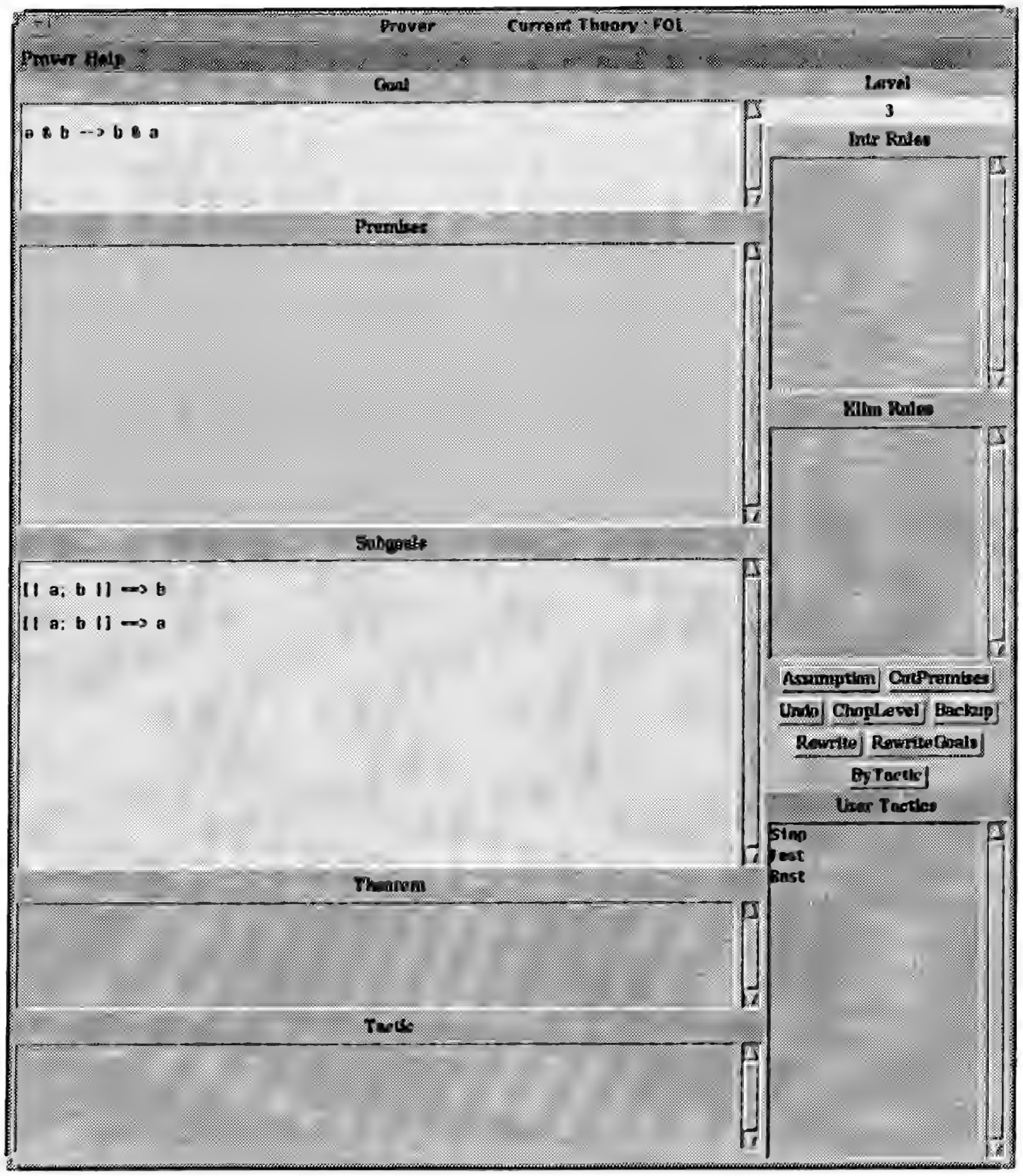
XIsabelle can handle goals which contain assumptions. For example, consider the following variation of the example proved above, where the object-level (FOL) implication has been replaced by meta-implication:

$$\frac{a \wedge b}{a \vee b}$$

This time the goal looks like this in the Term window:

```
  a & b ==> b & a
```

When we invoke the Prover, it will look like Figure 10. The assumption $a \wedge b$ is shown in the Premises window. It can be highlighted, and then the CutPremises button can be pressed. This has the effect of putting the assumption into the subgoal, and the proof then proceeds as before.



```
a & b --> b & a
```

```
3
```

```
[| a; b |] ==> b
[| a; b |] ==> a
```

```
Simp
Fast
Best
```

Figure 7: *Example Proof (cont)*

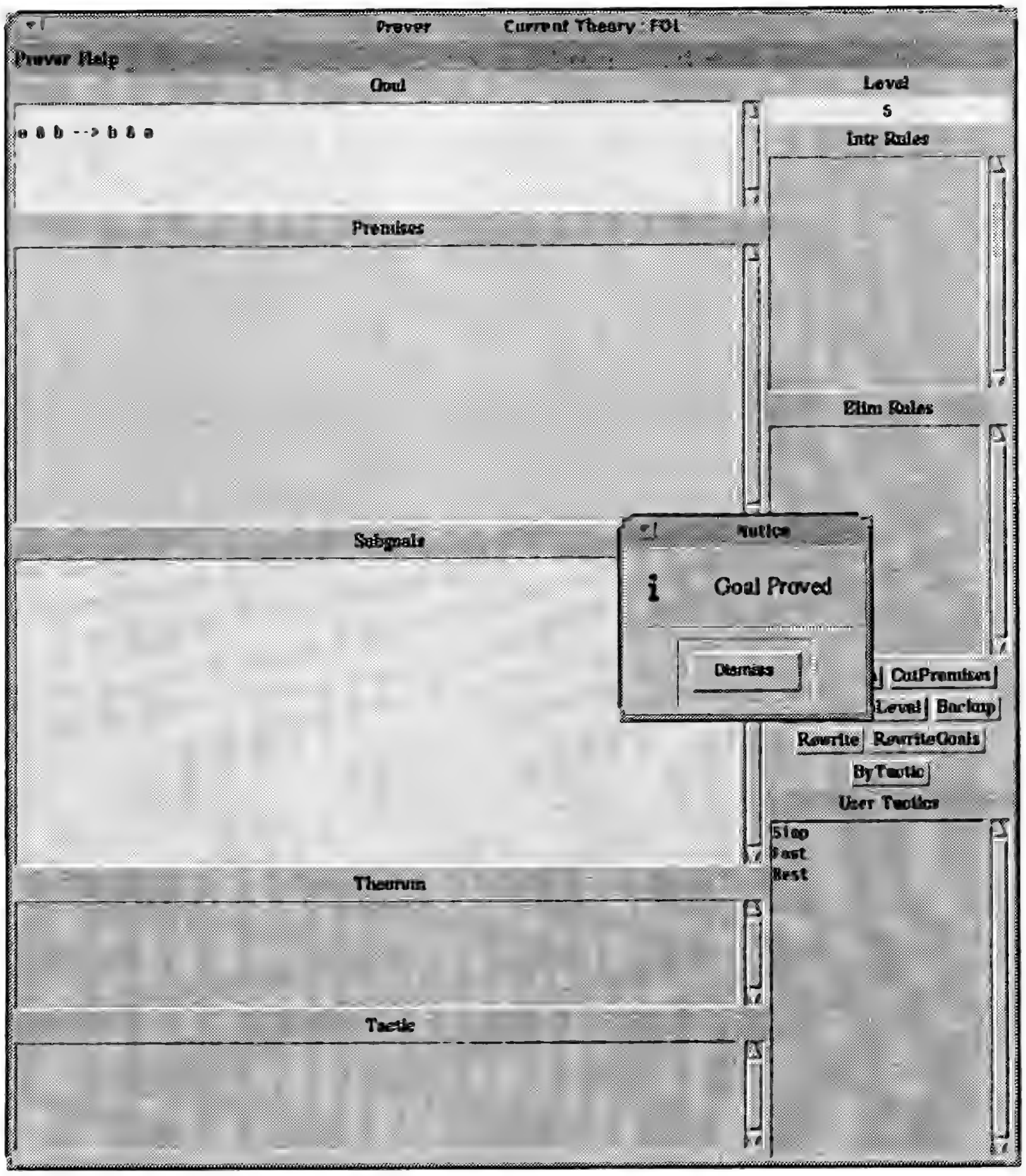
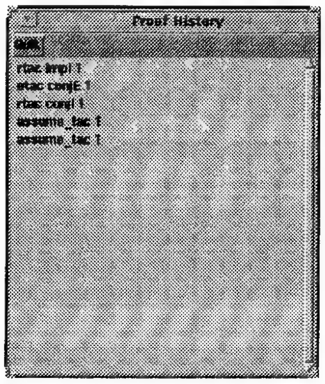
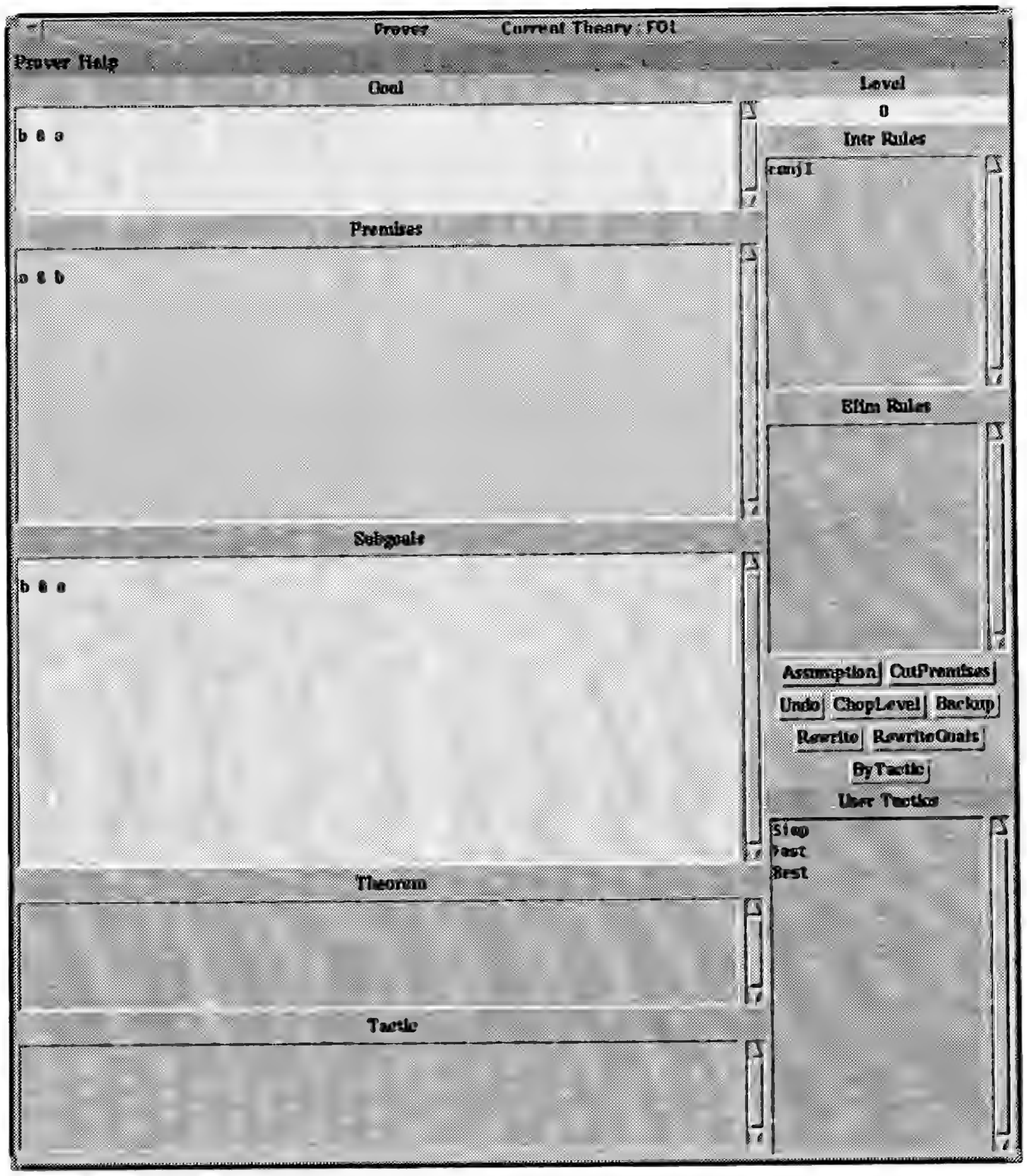


Figure 8: *Example Proof (cont)*



```
rtac impI 1
etac conjE 1
rtac conjI 1
assume_tac 1
assume_tac 1
```

Figure 9: *Example Proof History*



b & a

a & b

b & a

Figure 10: *Example Proof (cont)*

8 Conclusion

XIsabelle is still under development, and could be improved in a number of ways. The authors welcome any suggestions from users of the tool.

Our particular interest is in trustworthy, easy-to-use but powerful reasoning about the behaviour of critical systems. We are using XIsabelle as part of a more extensive reasoning environment which provides machine support for reasoning about the critical properties of such systems. In this work, XIsabelle will be used to provide support for both verifiers and engineers.

In this section, we outline some of the ways that XIsabelle could be enhanced.

8.1 Editing of Theories and ML Files

Syntax-directed editing is not supported by XIsabelle. This can be a very useful feature. Notable examples of tools which exploit it are `Mural` (for logical theories) and Balzac/Zola (for specifications in the Z language). Syntax-directed editing must be practical, and be able to be adjusted by the expert user, so that simple expressions do not take an excessive number of mouse operations to enter (the Balzac/Zola tool suffers from this defect, making the entering of Z specifications via their ASCII syntax the quickest method in pratice).

XIsabelle overloads the use of the file window, using it both for browsing existing theory or ML files, and for editing new ones. This was done to make the tool simpler to use. However, the editing facilities are fairly primitive, and there is a case for providing support for more powerful editing. Editors written in Tcl/Tk already exist (notably Don Libes' mxedit), and these could be easily invoked by XIsabelle for editing theories when required.

The file window currently provided with XIsabelle has very limited editing facilities: these are the basic Tk text widget operations including copy, cut and paste using the familiar keys on the Sun keyboard and the X windows middle-button paste as found in xterms. To make XIsabelle more useful for developing large theory or ML files, a better editor is needed. There are three possible ways to do this:

- provide more editing features in the file window;
- hook up XIsabelle to a separate Tcl/Tk based editor (such as mxedit);
- allow the user to specify an editor of choice, and start this editor in a shell window when required[11].

[11]For example, this is how editing is done in xtem, a Tcl/Tk-based graphical interface to LaTeX devised by G. Lamprecht, W. Lotz and R. Weibezahn at the University of Bremen.

8.2 Forwards Proof

XIsabelle does not provide any support for Isabelle's forwards proof, whereby new theorems produced from old ones by resolution, the use of `rule_by_tactic` etc. It is not clear what form such support should take. The Mural tool combines forwards and backwards proof techniques within one visual proof tool, and it would be worthwhile exploring this possibility in XIsabelle.

8.3 Backwards Proof

There are some deficiencies in the way XIsabelle handles backwards proof. The Backup command will correctly backup to an alternative proof state, if one exists. However, the Proof History is not correctly maintained in this case. The tool needs to be improved to support interactive searches over the proof tree.

Similarly, although Isabelle has the facility for tracing simplification steps, iterative proofs, tree searches etc, this has not been implemented in XIsabelle, although it probably would not be difficult to do so.

8.4 Theory Browsing

The structure of theories could be shown in a more informative way than is given in the theory file. For example, the constants of a theory could be displayed on request; the syntax translations shown etc. The Mural theorem prover is a good example of what can be done. The issue is whether the extra effort is worth it.

8.5 Information Hiding

It would be useful to be able to hide irrelevant information in the Prover by being able to collapse and expand large subgoals, thus enabling attention to be focused on the subgoal of interest without being distracted. The feature is found in the Balzac/Zola tool [24].

8.6 Final Comments

The authors are very conscious that XIsabelle is a tool which can grow in any number of possible ways. Valuable lessons can be learned from related tools as they are developed. Feedback on XIsabelle is welcomed from both experienced Isabelle users and beginners, because we have tried to please both.

To finish on a light-hearted note, we have found that carrying out theory and proof development using Isabelle usually requires using the 'spare-feet-on-the-desk paradigm' which has been stressed by Ritchie et al. for the Mural prover[12]. It can be very hard for one person to get through complex proofs alone; we have found that having one person 'drive' the proof, while another makes suggestions when needed, is an effective way of making progress. It has been our aim with XIsabelle (and we have not achieved this yet) to provide the extra pair of feet, and make it possible for a single user to do non-trivial proofs.

9 Acknowledgments

The authors wish to thank Jim Grundy for his help with the implementation, and Tony Dekker and Katherine Eastaughffe for many useful suggestions. Larry Paulson refereed the paper and provided useful comments. Thanks are due for his encouragement, and for making XIsabelle available from the Cambridge ftp site.

References

[1] D. Craigen, S. Gerhart, and T. Ralston. An International Survey of Industrial Applications of Formal Methods. *Volume 1*: Purpose, Approach, Analysis and Conclusions. *Volume 2*: Case Studies. Report, National Institute of Standards and Technology, Gaithersburg, MD 20899 USA, 1993.

[2] UK Ministry of Defence. *Interim Defence Standard 00–55: The Procurement of Safety-Critical Software in Defence Equipment*, April 1991.

[3] UK Ministry of Defence. *Draft Defence Standard 00–56: Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defence Equipment*, February 1993.

[4] Australian Ordnance Council, Department of Defence, Canberra. *Pillar Proceeding 223.93: Assessment of Munition Related Safety Critical Computing Systems*, August 1993.

[5] Department of Defense DoD 5200.28-STD. *Trusted Computer System Evaluation Criteria*, August 1985.

[6] Michael J. C. Gordon and Thomas F. Melham. *Introduction to HOL: A theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[7] L. C. Paulson. *Introduction to Isabelle, The Isabelle Reference Manual and Isabelle's Object Logics.* Computer Laboratory, University of Cambridge, 1995.

[8] D. Craigen. Strengths and Weaknesses of Program Verification Systems. Strasbourg, France, 9-11 September 1987. 1st European Software Engineering Conference, Springer-Verlag.

[9] A. Cant and M. A. Ozols. An Approach to Automated Reasoning About Operational Semantics. Research Report RR-0008, Electronics and Surveillance Research Laboratory, DSTO, 1994.

[10] W. A. Halang, B. Krämer, and N. Völker. Formally Verified Building Blocks in Functional Logic Diagrams for Emergency Shutdown System Design. *High Integrity Systems*, 1(3):277–286, 1995.

[11] D. Craigen et al. EVES: An Overview. Conference Paper CP-91–5402–43, Odyssey Research Associates, 265 Carling Avenue, Suite 506, Ottawa, Ontario, March 1991.

[12] C. B. Jones, K.L. Jones, P. A. Lindsay, and R. Moore. *Mural: A Formal Development Support System.* Springer-Verlag, 1991.

[13] D Clement. GIPE: Generation of Interactive Programming Environments. *Technique et Science Informatiques*, 9:157–165, 1990.

[14] L. Théry, Y. Bertot, and G. Kahn. Real Theorem Provers Deserve Real User-Interfaces. In *Proceedings of Fifth Symposium on Software Development Environments.* ACM, 1992.

[15] S. Finn and M. Crawley. *Using Poly/ML 2.05M.* Abstract Hardware Ltd, Building 2, The Science Park, Brunel University,Uxbridge, Middlesex.UB8 3PQ, U.K., October 1993.

[16] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.

[17] D. Libes. *Exploring Expect.* O'Reilly and Associates, 1995.

[18] D. Libes. X Wrappers for Non-Graphic Interactive Programs. In *Proceedings of Xhibition '94*, 1994.

[19] Lawrence C. Paulson and Tobias Nipkow. *Isabelle: A Generic Theorem Prover.* Springer Verlag, Lecture Notes in Computer Science Vol 828, 1994.

[20] L. C. Paulson. The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning*, 5:363–397, 1989.

[21] L. C. Paulson. Isabelle: The Next 700 Theorem Provers. *Logic and Computer Science (P Odifreddi, ed)*, pages 361–385, 1990.

[22] L. C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF.* Cambridge University Press, 1987.

[23] L. C. Paulson. A Fixedpoint Approach to Implementing (Co)Inductive Definitions. Report, University of Cambridge, 1994.

[24] Imperial Software Technology, Cambridge UK. *The Z User Manual (Volume I: User Guide and Tutorial, Volume II: Z Language Reference, Volume III: Programmer's Reference Manual)*, 1995.

Xisabelle: A Graphical User Interface
to the Isabelle Theorem Prover

A. Cant and M.A. Ozols

(DSTO-RR-0070)

DISTRIBUTION LIST

	Number of Copies
Defence Science and Technology Organisation	
Chief Defence Scientist and members of the)	1 shared copy
DSTO Central Office Executive)	for circulation
Counsellor, Defence Science, London	(Document Control sheet)
Counsellor, Defence Science, Washington	(Document Control sheet)
Senior Defence Scientific Adviser)	1 shared copy
Scientific Adviser - POLCOM)	
Assistant Secretary Science Industry Interaction	1
Director, Aeronautical & Maritime Research Laboratory	1
Navy Scientific Adviser (NSA)	1
Scientific Adviser, Army (SA-A)	1
Air Force Scientific Adviser (AFSA)	1
Electronics and Surveillance Research Laboratory	
Chief Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Executive Officer, Information Technology Division	(Document Control sheet)
Manager, Human Systems Integration Group	(Document Control sheet)
Head, Software Engineering Group	(Document Control sheet)
Head, Intelligence Systems Group	(Document Control sheet)
Head, Systems Simulation and Assessment Group	(Document Control sheet)
Head, Exercise Analysis Group	(Document Control sheet)
Head, C3I Systems Engineering Group	(Document Control sheet)
Head, Computer Systems Architecture Group	(Document Control sheet)
Head Information Management and Fusion	(Document Control sheet)
Head Command Support Systems Group	1
Head, Trusted Computer Systems Group	1
Head, Advanced Computer Capabilities Group	1
A. Cant (Author)	1
M.A. Ozols (Author)	1
Publications and Publicity Officer, ITD	1
Libraries and Information Services	
Defence Central Library - Technical Reports Centre	1

Manager Document Exchange Centre (MDEC) (for retention)	1
Additional copies which are to be sent through MDEC	
DIS for distribution:	
National Technical Information Centre. United States	2
Defence Research Information Centre, United Kingdom	2
Director Scientific Information Services, Canada	1
Ministry of Defence, New Zealand	1
National Library of Australia	1
Defence Science and Technology Organisation Salisbury, Research Library	2
Library Defence Signals Directorate Canberra	1
AGPS	1
British Library Document Supply Centre	1
Parliamentary Library of South Australia	1
The State Library of South Australia	1
XTP-1 National Leader United States	1
XTP-1 National Leader United Kingdom	1
XTP-1 National Leader Canada	1
XTP-1 National Leader New Zealand	1
Spares	
Defence Science and Technology Organisation Salisbury, Research Library	30

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. PRIVACY MARKING/CAVEAT (OF DOCUMENT) N/A
2. TITLE Xisabelle: A Graphical User Interface to the Isabelle Theorem Prover	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)
4. AUTHOR(S) A. Cant and M.A. Ozols	5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury SA 5108

6a. DSTO NUMBER DSTO-RR-0070	6b. AR NUMBER AR-009-478	6c. TYPE OF REPORT Research Report	7. DOCUMENT DATE December 1995

8. FILE NUMBER N9505/10/18	9. TASK NUMBER ADL 94/162	10. TASK SPONSOR DSD	11. NO. OF PAGES 54	12. NO. OF REFERENCES 24

13. DOWNGRADING/DELIMITING INSTRUCTIONS	14. RELEASE AUTHORITY Chief, Information Technology Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

APPROVED FOR PUBLIC RELEASE

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT
No limitation

17. CASUAL ANNOUNCEMENT No Limitation

18. DEFTEST DESCRIPTORS

Theorem proving
Graphical user interfaces
computer systems

19. ABSTRACT

Interactive theorem provers such as Isabelle are powerful tools, but are difficult and time-consuming to learn. If a suitable Graphical User Interface (GUI) is provided for such a tool, it can speed up the learning process considerably, leading to greater productivity for users of the tool, and increased takeup in industry. In this paper, we discuss the user-interface aspects of Isabelle, and formulate requirements for a GUI. XIsabelle, a GUI for Isabelle, is described in detail. XIsabelle uses standard, easily available, methods for providing X Windows wrappers to interactive non-GUI programs, namely Tcl/Tk and the program Expect.

Page classification: UNCLASSIFIED